RECEIVED
CENTRAL FAX CENTER

JUN 17 2005

I hereby certify that this correspondence is being transmitted via facsimile to the U.S. Patent Office at 703-872-9306 on the date shown below.

Dated: June 17, 2005 Signature: Christian R. Andersen

Docket No.: 00-VE20.57RCE1
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
Perushottam Yeluripati et al.

Application No.: 09/620,102 Art Unit: 2126

Filed: July 20, 2000 Examiner: ANYA, Charles E.

For: FUNCTIONAL ENTERPRISE BEAN

APPEAL BRIEF IN ACCORDANCE WITH 37 C.F.R. § 41.37

Mail Stop Appeal Brief- Patents
Commissioner for Patents
United States Patent and Trademark Office
Alexandria, VA 22313-1450

Dear Sir:

This appeal is from the decision of the Primary Examiner dated January 26, 2005 ("Final Office Action"), finally rejecting claims 1-19, which are reproduced as an Appendix to this Appeal Brief. The Notice of Appeal was filed on March 24, 2005. This application was filed on July 20, 2000. Submitted herewith are two additional copies of this Appeal Brief.

I. REAL PARTY IN INTEREST

The real party in interest is Telesector Resources Group, Inc., a Delaware corporation having a place of business at 1095 Avenue of the Americas, New York, New York, 10036.

II. RELATED APPEALS AND INTERFERENCES

Applicants (hereinafter "Appellants") are not aware of any related appeals or interferences that would affect the Board's decision on the current appeal.

06/20/2005 AKELECH1 00000033 09620102

01 FC:1402 500.00 DA

III. STATUS OF CLAIMS

Claims 1-19 are pending, and are the subject of this Appeal. In the Final Office Action, claims 1-6 were rejected under 35 U.S.C. § 103(a) as obvious over US 6,629,128 ("Glass") in view of US 6,510,550 ("Hightower"), and further in view Pospisil et al., On

Performance of Enterprise Java Beans ("Pospisil"). Claims 7-19 were rejected under 35 U.S.C. § 103(a) as obvious over Glass in view of Pospisil.

IV. STATUS OF AMENDMENTS

No Amendment After Final Rejection has been entered into the prosecution record of the present application.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Background:

By way of background, and for the discussion that follows, a reference to a "bean" indicates it is an Enterprise JavaBean (EJB) or a similar component. Additionally, an "application server," a "middleware server," an "enterprise application server," or a "server" are terms used interchangeably unless otherwise qualified, each of which terms indicates a computer connected to a communication network and at least one of a plurality of back-end computers or data repositories, for example, a legacy database server. (Specification, page 4, lines 17-22.)

The EJB is a server-side component model that incorporates business logic in software components called beans. An enterprise application server or simply, a "server," that is configured as an EJB server is responsible for managing services such as transactions, persistence, concurrence and security. The EJB architecture, which is a distributed object architecture, comprises one or more computers configured as EJB server(s), EJB container(s) that run within the EJB server(s), beans that execute within these containers, and EJB clients. (Specification, page 4, lines 23-29.)

Multiple EJB clients can share what are known as entity beans. Entity beans always have states and these states persist server shutdown or crash. Typically, entity beans are implemented by writing any state-related data into a persistent store so that the bean can be reconstructed from the stored data after a server is restarted subsequent to a shutdown or a crash. An entity bean instance is usually associated with a specific row (record) in a specific table of a database. More complex mappings are also possible. An entity bean is also associated with a transaction. The bean's state is always in step with the transaction-if the transaction commits, the bean's state persists, and if the transaction rolls back, the bean is brought back to its initial state. This synchronization is automatic and seamless to a developer, unless the bean is deployed as having bean-managed persistence, or no

transactional persistence at all. In general, a client can access a particular entity bean instance using an identifier or a key. (Specification, page 6, lines 7-17.)

Traditionally, sessions beans, i.e., beans that keep track of a user's session, and that do not survive a server shutdown or crash (Specification, page 5, lines 17-21), have been used to access entity beans. Thus, entity beans could be accessed only by a session bean, and not directly by a client application, according to the following model:

Client --> Session bean --> Entity bean --> Database Record.

This model is based on the idea that the entity bean must model only entities (i.e., one or more database records). However, if there is a need for a number of database records to be simultaneously accessed by a client, then there will be a proliferation of such beans if each record is modeled as an entity bean. In extreme cases, such a proliferation may result in a depletion of available resources for other tasks. Further, since access to these entity beans should be serialized, each successive client with a need to access a particular row (or an entity bean) for an update should wait for its turn. Moreover, the foregoing model implicitly assumes that the client knows beforehand the primary key (or the identifier) of the entity bean that it needs to contact. (Specification, page 6, line 28 – page 7, line 10.)

Claimed Subject Matter:

In view of the difficulties with accessing entity beans via the traditional model described above, the presently claimed subject matter achieves transactional synchronization as well as transactional integrity by using entity beans rather than session beans. Mapping an object to a function, rather than to a data element, provides significant advantages, especially in managing limited resources that are accessed by a plurality of clients. Further, the presently available EJB model can be programmed to achieve such a mapping. (Specification, page 7, lines 15-22.)

Thus, one claimed embodiment includes a novel "functional" bean that is devoted to modeling a business function. Clients do not need to know the particular primary key or identifier as in the case of an entity EJB; rather, a client knows only a well-known Service Manager bean to obtain a handle to the correct type of functional bean. If a client needs to request a service offered by a functional bean, it can be invoked directly. In one aspect, functional beans are created and managed by a bean container such as an EJB container. The container controls access to these beans by way of a Service Manager. In one claimed embodiment, the Service Manager itself is modeled as a functional bean, whose function is

that of a manager of the other functional beans. Thus, in one claimed embodiment, a three-tiered model that can accomplish the task contains the following structure.

Client --> Functional bean --> Database view/record

Further, there could be cases where a system does not require some attributes such as state management and persistence. In such a situation, a system designed using functional beans can be easily implemented without these attributes, typically associated with an entity EJB. (Specification, page 7, line 23 – page 8, line 6.)

In certain claimed embodiments, the characteristics of a functional bean may include:

a) Transactional awareness, which is a default characteristic in an embodiment and is implemented by providing a persistent database to which transactional data are written.

b) Optionally, transparent support of transactional persistence, i.e., the data objects operated on by a functional bean's methods and any state information are persisted.

c) Allowing multiple clients to use a single instance of a functional bean. Since functional beans are designed to incorporate functions rather than data, they can receive a client request from a queue and service the client request without changing the request's own internal state. These requests can be serviced according to a particular scheme, such as first-come, first-serve, and the like. In alternative embodiments, the functional bean can service client requests using a priority method as designed by a programmer or others skilled in the art.

d) A pool of functional beans of each functional type is maintained by the enterprise application server to load-balance the requests from clients.

e) Load sharing-handled according to a strategy that is either static or dynamic, such as round robin or least busy. A client need not know which instance of each type of functional bean is in communication with the client at any given moment; the client is required only to be aware of the type of the bean with which it is communicating.

f) The number of functional beans of each type in the pool can be configured at startup and/or dynamically controlled by the bean's container. Depending on the number and availability of limited resources-for example, the number of open connections allowed by a legacy system-determines the number of bean instances

of each type. In claimed embodiment, a system administrator makes an appropriate choice of the number of instances of each bean type either statically or dynamically.

h) The functional bean does not map to one or more rows in a database.

i) A functional bean's methods do not by default allow for data persistence. An application or a business-logic programmer may choose to provide for explicit data persistence by programming the appropriate code, for example, by providing appropriate JDBC statements.

(Specification, page 10, line 17 – page 11, line 15.)

VI. GROUNDS OF REJECTION

1. Claims 1-6 were rejected under 35 U.S.C. § 103(a) as obvious over US 6,629,128 ("Glass") in view of US 6,510,550 ("Hightower"), and further in view Pospisil et al., On Performance of Enterprise Java Beans ("Pospisil").

2. Claims 7-19 were rejected under 35 U.S.C. § 103(a) as obvious over Glass in view of Pospisil.

The issues are:

1. Whether the combination of Glass, Hightower, and Pospisil teaches "returning . . . a handle to a functional bean" as required by claim 1.

2. Whether the combination of Glass, Hightower, and Pospisil teaches "a data store interface for coupling said application service program to a data storage system . . ." as required by claim 1.

3. Whether the combination of Glass, Hightower, and Pospisil teaches "memory . . . containing instructions. . . for servicing the queued customer requests in accordance with the code contained in the functional bean . . ." as required by claim 1.

4. Whether the combination of Glass, Hightower, and Pospisil teaches "the functional bean is configured to provide transactional persistence to a client transaction" as required by claim 4.

5. Whether the combination of Glass and Pospisil teaches "functional beans" as required by claims 7 and 12.

6. Whether the combination of Glass and Pospisil teaches "a load-sharing program . . .", "instructions to create a number of instances of functional beans . . .", and "provid[ing] transactional access to a pool of scarce system resources" as required by claims 7, 12, and 18.

7. Whether the combination of Glass and Pospisil teaches "deriving a class with no data elements . . ." as required by claim 18.

VII. ARGUMENT

A. Claims 1-6 Are Patentable Over The Proposed Combination of Glass, Hightower, and Pospisil.

Claims 1-6 stand rejected as obvious over Glass in view of Hightower and Pospisil. Independent claim 1, from which claims 2-6 depend, recites as follows:

> A computer system for supporting communication between a plurality of users and at least one application server comprising:
> an application service program on the at least one application server for receiving a client request from a client program executing on a computer associated with at least one of the users;
> a client interface program for communicating messages between said client program and said application service program;
> a service manager bean coupled to said application service program for creating and returning to said client program a handle to a functional bean appropriate to the client request wherein the functional bean is configured to model a business function;
> a data store interface for coupling said application service program to a data storage system, said data storage system comprising at least one computing device, wherein said computing device is not said computer associated with said at least one of the users; and
> memory coupled to said application service program, said memory for queuing customer requests and for servicing the queued customer requests in accordance with the code contained in the functional bean, said code providing for access to the data storage system via the data store interface.

The Examiner has failed to state a *prima facie* case of obviousness with respect to claim 1 at least because (1) Glass, Hightower, and Pospisil, even when combined, fail to teach or

suggest each and every one of the limitations of Appellants' claims, and (2) there is no teaching or suggestion in the prior art references to combine them to achieve the claimed invention. *See* MPEP § 2143. Accordingly, Appellants respectfully urge this Board to reverse the rejections of claim 1-6 for the independent reasons set forth below.

1. Claim 1: "returning . . . a handle to a functional bean"

Claim 1 recites, *inter alia*, "returning to said client program a handle to a functional bean appropriate to the client request, wherein the functional bean is configured to model a business function." The Examiner asserted that Glass teaches this claim limitation. (Final Office Action, pages 2-3.) However, while Glass discloses "specialized function objects referred to as EJB function objects" (Glass, col. 15, lines 38-40), despite the similarity in nomenclature nothing in Glass teaches or suggests a "functional bean," as is required by claim 1.

Glass is directed toward dynamically (*i.e.*, at run-time) generating proxy classes that support communications between client applications and server objects. (Glass, col. 4, lines 5-8.) Glass also teaches generating a number of other objects on both a client and a server to support client-server communications. (Glass, col. 4, lines 10-38.) Among these dynamically-generated objects are "type objects" for enabling access to the methods of a server object (Glass, col. 4, lines 7-8, 33-34), and "function objects", which provide a connection and correspond in number to the methods of a server object. (Glass, col. 4, lines 16-18, 34-38.) Thus, among the functional objects disclosed by Glass are EJB function objects. Each EJB function object resides on the server and corresponds to a method in a server object. (Glass, Figs. 9 and 10.) The purpose of EJB function objects is to replace the need to create wrapper classes for each object on a server by providing "a common class, EJB function." (Glass, col. 15, lines 1-6.)

The above-described disclosure of Glass wholly fails to disclose any element corresponding to the recited "functional bean" in the context of the claimed invention. The function objects and EJB function objects disclosed by Glass do no more than "preliminary processing" common to many server objects before invoking a method in a server object. (Glass, col. 16, lines 8-15.) Glass contains no teaching or suggestion that EJB function objects contain any business logic. In fact, function objects, including EJB function objects, in corresponding to a single method of a server object having no relation to any particular business function, clearly teach away from functional beans that model an entire business

function using a plurality of methods to implement business logic unique to that business function. (*See, e.g.,* Specification, page 15, lines 11-29.)

Further, the "type object" disclosed by Glass contains the set of function objects that have a one-to-one correspondence with the methods in an associated server object. (Glass, col. 14, lines 29-33.) Type objects are in no way analogous to a "functional bean" because they fulfill no more than the function of forwarding messages to the appropriate function object, and clearly do not model or in any way represent business functions. (Glass, col. 16, lines 8-10.) Indeed, the purpose of Glass is to dynamically generate proxy classes and other classes to support client-server communications, but unlike Appellants' disclosure as reflected in claim 1, Glass contains no teaching or suggestion of altering the conventional Enterprise Java Beans architecture to model business functions in a functional bean accessible from a client. Therefore, since Glass does not teach a functional bean, it cannot teach or suggest a *handle* to a functional bean and it therefore cannot teach or suggest "returning to said client program a handle to a functional bean appropriate to the client request, wherein the functional bean is configured to model a business function," as required by claim 1.

In sum, Glass is wholly silent with respect modeling business functions, and contains no teaching or suggestion of "returning a handle to a functional bean . . . configured to model a business function." Accordingly, at least for the foregoing independent reasons, the 35 U.S.C. § 103(a) rejection of claim 1, as well as of claims 2-6 depending therefrom, is improper, and should be reversed.

2. Claim 1: "a data store interface . . ." and "memory . . . containing instructions. . . for servicing the queued customer requests in accordance with the code contained in the functional bean . . ."

Claim 1 recites "a data store interface for coupling said application service program to a data storage system, said data storage system comprising at least one computing device, wherein said computing device is not said computer associated with said at least one of the users." Claim 1 further recites "memory coupled to said application service program containing instructions executable by a processor . . . for servicing the queued customer requests in accordance with the code contained in the functional bean, said code providing for access to the data storage system via the data store interface." The Examiner acknowledged that Glass does not teach these claim limitations, but contended that they would have been obvious over Glass in view of Hightower and Pospisil. (Final Office Action, page 3).

However, even assuming that Hightower and Pospisil taught the above-quoted claim limitations, which, as discussed below, they do not, neither reference contains a motivation for one of ordinary skill in the art to have combined the reference with the teachings of Glass, as also discussed below.

a. Hightower

The Examiner contended (Final Office Action, page 3) that Hightower teaches "memory coupled to said application service program containing instructions executable by a processor . . . for servicing the queued customer requests in accordance with the code contained in the functional bean, said code providing for access to the data storage system via the data store interface," as is required by claim 1. In fact, Hightower is of little relevance to claim 1. Hightower teaches "a method . . . for providing an application with intermittent connectivity support." (Hightower, Abstract.) Accordingly, Hightower teaches storing method calls from a client to a server in a data store *on the client* until such time as there is a connection between the client and the server. (Hightower, col. 8, lines 27-42.) As the Examiner noted (Final Office Action, page 3), Hightower provides the benefit of allowing the user of a client application continuous use of the application "even in the absence of a connection to [an] enterprise application." (Hightower, col. 8, lines 42-45.) However, Hightower contains no teaching or suggestion to use code in a functional bean providing access to a data storage system as is required by claim 1.

(1) Failure Of Glass and Hightower To Teach Each and Every Recited Claim Limitation

Initially, the Examiner's rejection of claim 1 as obvious over Glass in view of Hightower was improper because the references simply do not teach each and every recited claim limitation. In particular, Hightower cannot read on claim 1 because Hightower's data store is resident on a client computer, thus eliminating any need for a "memory coupled to said application service program . . ." as is required by claim 1. Indeed, in the context of claim 1, both the recited "data store interface" and "data storage system" are *not* on a client. Rather, the claimed data store interface exists for coupling an application service program, explicitly recited to be on a server, to a data storage system that is explicitly recited to be on a machine other than a client. Hightower, in contrast, discloses an entirely different architecture in which a data store is located on a client, and in which, therefore, a memory coupled to a program on a server for queuing requests would make no sense, and indeed, would plainly be inoperable. Thus, Hightower plainly cannot teach or suggest "*memory*

coupled to said application service program, said memory for queuing customer requests and for servicing the queued customer requests in accordance with the code contained in the functional bean, said code providing for access to the *data storage system* via *the data store interface*" (emphases added).

(2) Inability To Combine Glass and Hightower To Achieve The Limitations Of Claim 1

Because Hightower locates its data store on the client, any modification of Glass with the teachings of Hightower would result in a system with substantial structural differences from Appellants' claimed invention. Again, claim 1 requires "a data store interface for coupling said application service program to a data storage system, said data storage system comprising at least one computing device, *wherein said computing device is not said computer associated with said at least one of the users*" (emphasis added). Assuming *arguendo* that Glass and Hightower even could be combined, such combination would result in Glass' server system 12 accessing a data store on Glass' client 14. (*See* Glass, Fig. 1.) Claim 1, however, explicitly excludes the structure that would result from the alleged combination of Glass and Hightower because claim 1 explicitly provides that "said data storage system" cannot comprise "said computer associated with said at least one of the users." Therefore, Glass and Hightower are incapable of combination to meet the limitations of claim 1, and the rejection of claim 1 over the combination of Glass and Hightower is improper for this reason alone.

(3) Lack of Motivation To Combine Glass and Hightower

Further, assuming *arguendo* that Glass discloses a functional bean, which it does not, Hightower contains no disclosure that would have motivated one of ordinary skill in the art to modify Glass with a data storage interface and "memory coupled to said application service program containing instructions executable by a processor . . . for servicing the queued customer requests in accordance with the code contained in the functional bean, said code providing for access to the data storage system via the data store interface," as is required by claim 1. Hightower discloses at most the need to provide "applications with the ability to support intermittent connectivity processing." (Hightower, col. 2, lines 17-19.) Appellants, in contrast, address the problem of the depletion of server resources caused by the proliferation of server objects such as entity beans. (Specification, page 7, lines 3-14.)

Hightower's solution to its disclosed need for supporting intermittent connectivity processing is to place a data store on client machines running applications requiring such

support. Thus, both Hightower's disclosed problem, and Hightower's solution to that problem, have nothing to do with enabling access to a database server from a server object such as a functional bean. Hightower's stated need for supporting intermittent connectivity processing on a client could not have motivated one of ordinary skill to implement a "memory coupled to said application service program" on a server, which is what claim 1 explicitly requires. Similarly, Hightower simply cannot contain any teaching or suggestion that would have motivated one of ordinary skill in the art to implement the recited "data store interface" or "functional bean [having] code providing for access to the data storage system via the data store interface." The rejection of claim 1 over the combination of Glass and Hightower is improper for this reason alone.

b. Pospisil

The Examiner contended (Final Office Action, page 3) that Pospisil teaches "a data store interface for coupling said application service program to a data storage system," as is required by claim 1. The Examiner further contended that it would have been obvious to combine Pospisil and Glass "because the teaching of Pospisil would improve the system of Glass by providing database updating." However, even if Pospisil taught the afore-mentioned claim limitation, the general statement of motivation that the Examiner has attributed to Pospisil is insufficient for one of ordinary skill in the art to have modified Glass.

Pospisil (page 3) teaches no more than that Enterprise Java Beans "execute . . . database updates using the standard JDBC API." Pospisil contains no teaching or suggestion of functional beans, much less of using functional beans to provide for access to a data storage system via a data store interface. The mere fact that Pospisil generally discusses database transactions such as updating in no way would have provided specific motivation to use functional beans to provide for access to a data storage system via a data storage interface. Indeed, Appellants' claimed invention addresses not the need to provide for database transactions such as updating, but rather to provide for such transactions in a more efficient way, *i.e.*, by using functional beans. Nothing in Pospisil suggests any problem or need arising from Enterprise Java Beans as known in the prior art that would have motivated one of ordinary skill to have implemented functional beans providing for access to a data storage system as recited in claim 1. Furthermore, Pospisil does not cure any of the deficiencies noted above with respect to Glass and Hightower.

At least for the foregoing reasons, claim 1, as well as claims 2-6 depending therefrom, is patentable over the combination of Glass, Hightower, and Pospisil.

3. Claim 4: ". . . transactional persistence . . ."

Claim 4 recites that "the functional bean is configured to provide transactional persistence to a client transaction." The Examiner asserted (Final Office Action, page 4) that Glass teaches this claim limitation. However, the cited portion of Glass discloses only that the EJB function object provides "preliminary processing" including "transaction management." (Glass, col. 16, lines 10-15.) Glass says nothing at all about transactional *persistence*. Appellants' Specification (page 10, lines 21-22) explains that "transactional persistence" means that "the data objects operated on by a functional bean's methods and any state information are persisted." Glass contains absolutely no discussion of persisting either data objects or state information of any kind. Accordingly, Glass does not teach or suggest that "the functional bean is configured to provide transactional persistence." Therefore, for at least the foregoing additional independent reason, claim 4 is separately patentable.

B. Claims 7-19 Are Patentable Over The Proposed Combination of Glass and Pospisil.

Claims 7-19 stand rejected over Glass in view of Pospisil. However, the Examiner has failed to state a *prima facie* case of obviousness at least because (1) Glass and Pospisil, even when combined, failed to teach or suggest each and every one of the limitations of Appellants' claims, and (2) there is no teaching or suggesting in the prior art references to combine them to achieve the claimed invention. *See* MPEP § 2143. Accordingly, Appellants respectfully urge this Board to reverse the rejections of claims 7-19 for each of the independent reasons set forth below.

1. Claims 7 and 10: "functional beans"

Claim 7 recites, *inter alia,* "a plurality of sets of functional beans, each set comprising at least one functional bean assigned to perform a particular business method" and also recites that "the client is configured to use the handle to interact with the functional bean to execute a business method." Claim 10 recites, *inter alia,* a "functional bean, said functional bean comprising code to execute a particular business function." Accordingly, claims 7 and 10 are in condition for allowance at least for the reasons stated above with respect to claim 1 regarding Glass' failure to teach or suggest functional beans. Further, claims 8-9, depending from claim 7, and claims 11-17, depending from claim 10, are also in condition for allowance for at least the reasons stated above.

2. Claims 7, 12 and 18: "a load-sharing program . . .", "instructions to create a number of instances of functional beans . . .", and "provid[ing] transactional access to a pool of scarce system resources"

Claim 7 recites "a load-sharing program coupled to the service manager program and configured to create instances of functional beans based on a criterion." Claim 12 recites the code of a functional bean "further comprising instructions to create a number of instances of functional beans of the particular type requested, said number being dependent on available of resources." Claim 18 recites instructions in a functional bean that "provide transactional access to a pool of scarce system resources." The Examiner (Final Office Action, page 5) acknowledged that Glass is silent with respect to these claim limitations, but asserted that they would have been obvious in view of Pospisil because "the teaching of Pospisil would improve the system of Glass by optimizing resource usage." (Office Action, pages 6, 7, 8.)

In fact, Pospisil (page 7) teaches no more than that "[s]calability is one of the most important performance aspects" of an Enterprise Java Beans application. Accordingly, Pospisil teaches a number of factors to test when evaluating an application's performance. However, Pospisil contains no teaching or suggestion of creating functional beans – or any kind of object, for that matter – based on a criterion. Nor does Pospisil contain any teaching or suggestion of instructions in a functional bean, much less instructions for "creating a number of instances of functional beans" or that "provide transactional access to a scarce pool of system resources." Moreover, Pospisil contains no disclosure that would have motivated one of ordinary skill in the art to implement a load-sharing program as is required by claim 7, instructions regarding the number of instances of functional beans created as is required by claim 12, or instructions providing transactional access to a pool of scarce system resources as is required by claim 18.

Therefore, claims 7, 12, and 18 are in condition for allowance for at least two independent reasons. First, the cited references, including Glass and Pospisil, fail to teach all of the limitations of claims 7, 12, and 18. Second, even if Pospisil did teach these limitations, the cited references contain no motivation to modify Glass with the recited load-sharing program or instructions in a functional bean. Further, claims 8-9, depending from claim 7, and claim 19, depending from claim 18, are also in condition for at least each of these independent reasons.

3. Claim 18: "deriving a class with no data elements . . ."

Claim 18 recites in part "deriving a class with no data elements from the object-oriented middleware component." The Examiner asserted that Glass teaches this claim limitation. (Final Office Action, pages 7-8.) However, Glass contains absolutely no teaching or suggestion of "deriving a class" as is required by claim 18. Glass teaches merely that "[u]nique functionality may be added to each EJBfunction object after it has been instantiated to provide for unique processing needs included in function object." (Glass, col. 18: 56-59.) The Examiner apparently relied on this statement to meet the limitation in claim 18 of "adding a set of computer-executable instructions to the derived class", in which case Glass cannot also be disclosing "deriving a class" as is separately recited in claim 18. In any event, Glass contains absolutely no disclosure of what this unique functionality is or how it is added, and therefore cannot be said to teach or suggest "deriving a class."

Further, Glass contains absolutely no teaching or suggestion that its EJB function object possesses no data elements and therefore Glass cannot read on "deriving a class with no data elements" for this independent reason.

For at least the foregoing reasons, claim 18 is in condition for allowance, as is claim 19, depending therefrom.

CONCLUSION

In view of the foregoing arguments, Appellants respectfully submit that the pending claims are novel over the cited references. The Examiner's rejection of Claims 1-19 is improper because the prior art of record does not teach or suggest each and every element of the claimed invention. In view of the above analysis, a reversal of the rejections of record is respectfully requested of this Honorable Board.

Appellants believe that no fee is due with this response. However, if a fee is due, please charge our Deposit Account No. 07-2347, under Order No. 00-VE20.57, from which the undersigned is authorized to draw. To the extent necessary, a petition for extension of time under 37 C.F.R. § 1.136 is hereby made, the fee for which should be charged to the above account.

Respectfully submitted,

Application No.: 09/620,102 Docket No.: 00-VE20.57RCE1

Date: June 17, 2005 By: _____

Joel Wall, Reg. No. 25,648
Attorney for Applicants
Verizon Corporate Services Group, Inc.
c/o Christian Andersen
600 Hidden Ridge
Mailcode HQE03H14
Irving, TX 75038
972-718-4800
Customer No. 32127

APPENDIX – CLAIMS ON APPEAL

1. A computer system for supporting communication between a plurality of users and at least one application server comprising:

an application service program on the at least one application server for receiving a client request from a client program executing on a computer associated with at least one of the users;

a client interface program for communicating messages between said client program and said application service program;

a service manager bean coupled to said application service program for creating and returning to said client program a handle to a functional bean appropriate to the client request wherein the functional bean is configured to model a business function;

a data store interface for coupling said application service program to a data storage system, said data storage system comprising at least one computing device, wherein said computing device is not said computer associated with said at least one of the users; and

memory coupled to said application service program, said memory for queuing customer requests and for servicing the queued customer requests in accordance with the code contained in the functional bean, said code providing for access to the data storage system via the data store interface.

2. The computer system of claim 1 wherein the functional bean is accessible by a program running on the client via an EJBObject.

3. The computer system of claim 1 wherein the functional bean is a modified entity bean.

4. The computer system of claim 1 wherein the functional bean is configured to provide transactional persistence to a client transaction.

5. The computer system of claim 1 wherein the client is web-based.

6. The computer system of claim 1 wherein the client is an application, an applet, a servlet or a JSP that can communicate with an EJF Object's remote interface using RMI over TCP/IP or IOP.

7. A computer system comprising a plurality of sets of functional beans, each set comprising at least one functional bean assigned to perform a particular business method, the computer system comprising:

a microprocessor;

a memory device coupled to the microprocessor;

a service manager program coupled to the memory device and configured to receive a number of requests from at least one of a plurality of types of transactions from a plurality of clients;

at least one of a plurality of resources;

a load-sharing program coupled to the service manager program and configured to create instances of functional beans based on a criterion;

the service manager program configured to obtain a handle to an instance of a functional bean based on a type of transaction requested by a client; and

the service manager program configured to return the handle to the client, wherein the client is configured to use the handle to interact with the functional bean to execute a business method, wherein the functional bean is configured to model a business function.

8. The computer system of claim 7 wherein the criterion in based on the number of resources.

9. The computer system of claim 7 wherein the criterion is based on the number of requests from the plurality of clients.

10. Computer-processor-executable software code stored in a computer-readable memory, said code comprising:

instructions to direct a computer processor to receive a first request from a client for a handle to a particular type of functional bean;

instructions to direct the computer processor to create an instance of a functional bean of the particular type requested;

instructions to direct the computer processor to obtain a handle to said instance of said functional bean; and

instructions to direct the computer processor to transmit to the client said handle to said instance of said functional bean,

wherein the computer processor, responsive to a second request from the client enables the client to execute code comprised in the functional bean to accomplish a particular business function, wherein the functional bean is configured to model the business function.

11. The code of claim 10, further comprising instructions to receive the first request and the second request from the client via a computer network.

12. The code of claim 10, further comprising instructions to create a number of instances of functional beans of the particular type requested, said number being dependent on available of resources.

13. The code of claim 10, further comprising instructions that allow a functional bean to instantiate a second functional bean of a second type in order to execute the business logic contained in the second functional bean instance.

14. The code of claim 10, further comprising:

instructions that allow a client to create a session with an instance of a session Enterprise JavaBean; and

instructions that allow the session Enterprise JavaBean to provide access to invoke the business methods contained in the functional bean.

15. The code of claim 10, further comprising:

instructions to instantiate an entity Enterprise JavaBean, said entity Enterprise JavaBean containing logic that maps a particular entity, and methods to perform actions on the particular entity; and

instructions to invoke the methods contained in the entity Enterprise JavaBean from the business methods contained in the functional bean.

16. The code of claim 10, wherein the instructions to create an instance of a functional bean of the particular type requested further comprise:

instructions to verify if required system resources are available.

17. The code of claim 16 further comprising:

if required system resources are not available, instructions to direct the computer processor to an existing instance of a functional bean of the particular type requested.

18. A method of using an object-oriented middleware component to create a functional bean, the method comprising the steps of:

deriving a class with no data elements from the object-oriented middleware component; and

adding a set of computer-executable instructions to the derived class,

wherein said set of computer-executable instructions is configured to provide transactional access to a pool of scarce system resources allowing client requests to be queued on EJB instances taken from the pool.

19. The method of claim 18, wherein the object-oriented middleware component is an entity Enterprise JavaBean.